VŠB– Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Simulace slovníkových kompresních metod pomocí open source nástroje Octave

Simulation of Compression Methods with Open Source Project Octave

2016 **Samuel Kavaler**

VSB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání bakalářské práce

Student: **Samuel Kavaler**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R059 Mobilní technologie

Téma: Simulace slovníkových kompresních metod pomocí open source nástroje Octave
Simulation of Compression Methods with Open Source Project Octave

Jazyk vypracování: čeština

Zásady pro vypracování:

Kompresní metody jsou využívány v komunikacích pro zvýšení přenosové rychlosti a jsou také využívány při šifrování dat. Cílem bakalářské práce je simulace slovníkových kompresních metod v prostředí open-source nástroje Octave.

1. Studium a popis slovníkových kompresních metod.
2. Studium a popis simulační prostředí Octave.
3. Simulace komprimace dat pomocí slovníkových metod.
4. Simulace dekomprimace dat pomocí slovníkových metod.

Seznam doporučené odborné literatury:

[1]Khalid Sayood *Introduction to Data Compression*. Morgan Kaufmann; 4 edition 2012, ISBN 978-0124157965

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Pavel Nevlud**

Datum zadání: 01.09.2015
Datum odevzdání: 29.04.2016

doc. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prehlásenie študenta

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave dňa: 29.4.2016

.....
podpis študenta

Poďakovanie

Rád by som poďakoval Ing. Pavlovi Nevludovi za rady, pripomienky a odpobné vedenie pri vypracovaní mojej bakalárskej práce.

Abstrakt

Bakalárska práca je zameraná na kompresné metódy a ich simuláciu v open source nástroji GNU Octave. Konkrétne pojednáva o slovníkových kompresných metódach. Obsah tvorí popis a vysvetlenie princípu kompresných metód, objasnenie hlavných pojmov, základné rozdelenie, bližší popis slovníkových kompresných metód, stručný popis inštalácie GNU Octave, popis simulačného prostredia Octave, jeho výhody a nevýhody, operácie s balíčkami, približuje prácu so skriptami a m-súbormi v GNU Octave a obsahuje simulácie kompresie a dekompresie dát.

Cieľom bolo zoznámiť sa s kompresnými metódami so zameraním na slovníkové kompresné metódy, priblížiť si open source nástroj Octave a nasimulovať si kompresiu a dekompresiu niektorých z vysvetlených metód.

Kľúčové slová

Octave, kompresia, dekompresia, kompresná metóda, slovník, slovníková kompresná metóda, simulácia

Abstract

My Bachelor thesis is focused on compressing methods and their simulation in open source tool GNU Octave. Specifically it's about dictionary compressing methods. This thesis consists of description and explanation of principle of compressing methods, clarifying main therms, basic distribution, closer description of dictionary compressing methods, brief description of installation of GNU Octave, description of simulating environment Octave, the pros and cons, operations with packages, closes up work with scripts and files in GNU Octave and contains simulations of compression and decompression of data.

The point was to inform about compressing methods with focus on dictionary compressing methods, close up open source tool Octave and simulate compression and decompression of some explained methods.

Key Words

Octave, compression, decompression, compressing method, dictionary, dictionary compressing method, simulation

Obsah

Zoznam použitých symbolov a skratiek

Skratka	Význam
A	Prehliadacie okno
B	Aktuálne okno
CLI	Command-line – príkazový riadok
DCT	Discrete cosine transform – diskrétna kosinusová transformácia
GNU	GNU's Not Unix! – GNU nie je UNIX!
GUI	Graphical user interface – grafické uživateľské rozhranie
M	Reťazec znakov
MS	Microsoft
OS	Operating system – operačný systém
RLE	Run-length encoding
a	znak za nájdenou frázou
i	vzdialenosť v počte znakov
j	dĺžka reťazca v aktuálnom okne
z	znak za reťazcom v aktuálnom okne

Zoznam ilustrácií a zoznam tabuliek

Zoznam výpisov z Octave a zdrojových kódov

Úvod

Cieľom bakalárskej práce je pochopenie základných slovníkových kompresných metód a osvojenie si práce v simulačnom prostredí programu GNU Octave, v ktorom budeme simulovať kompresiu a dekompresiu dát. Práca je štrukturovaná do niekoľkých hlavných kapitol.

Prvá kapitola nás uvádza do histórie a základou kompresie, oboznamuje nás so základnými pojmami, delením a typmi kompresií.

Druhá kapitola sa zaoberá konkrétne slovníkovými kompresnými metódami. Vybral som metódy LZ77, LZSS, LZ78 a LZW.

Tretia kapitola nám predstavuje simulačné prostredie GNU Octave, jeho výhody a nevýhody, stručne popisuje inštaláciu programu, popisuje jeho grafické prostredie, operácie s balíčkami, približuje prácu so skriptami a m-súbormi.

Posledná štvrtá kapitola obsahuje simulácie kompresie a dekompresie niektorých z vysvetlených slovníkových kompresných metód v prostredí Octave.

V elektronickej prílohe prikladám súbory s funkciami, s ktorými som pracoval.

1. Kompresné metódy

1.1 História

Človek od nepamäti potrebuje svoje veci zmenšovať a odľahčovať (skomprimovať). Stále má problém s prenosom vecí (dát) na úkor miesta.

Už v začiatkoch počítačov sa začala používať kompresia dát, no z počiatku nebola moc výkonná ani rýchlá a veľmi sa pri nej zahrieval procesor. Táto nevýhoda sa však dokázala využiť k vyhrievaniu priestorov kancelárií. V každom počítači bol jednoduchý program MS Office Heating, ktorý stále dookola spúšťal kompresiu, čo bol lacný spôsob výroby tepla.

Kompresia sa rozšírila až s rozvojom internetu a dnes ju používa každý užívateľ počítača.

1.2 Úvod do kompresie

Kompresia prebieha pomocou komprimačných programov, ktorých funkciou je zmenšiť veľkosť dátových súborov na pevnom disku. Dôležité je, že rôzne druhy dát, potrebujú rozdielny prístup k ich komprimácii, takže musíme rozdielne komprimovať súbory textové, binárne, tie čo obsahujú obrázky, fotografie a videá.

Medzi hlavné parametre, ktoré nás pri kompresii zaujímajú patrí: rýchlosť kompresie, rýchlosť dekompresie a kompresný pomer.

Kompresia, alebo komprimácia je metóda, pri ktorej dochádza k zmenšovaniu objemu dát tak, aby zaberal menší priestor. Je to teda výhodný spôsob ako si môžeme ušetriť čas a prenášať menšie množstvá dát.

Pojem kompresia nepatrí len medzi termíny z informatiky, so zmenšovaním objemu dát sa stretávame napríklad aj vo fyzike, kde sa jedná a zmenšovanie objemov plynu či v geológii.

1.3 Základné pojmy

Kompresia dát - je špeciálny prípad kódovania, s cieľom zmenšenia objemu dát odstránením nadbytočnej informácie, kedy je možné tieto dáta obnoviť v plnom rozsahu.

Komprimácia dát - je postup snažiaci sa postihnúť charakter dát a eliminovať ich dátovu redundanciu.

Redundancia - obecne označuje taký stav alebo vlastnosť, kedy je použité väčšie množstvo prvkov ako je obvyklé alebo nevyhnutné.

Kompresný pomer - je pomer objemu komprimovaných dát k objemu dát nekomprimovaných.

Fráza - je označenie používané v teórii slovníkových metód pre opakujúce sa časti textu.

Kompresná metóda a kompresný princíp (kódovanie)

Existuje mnoho rôznych kompresných metód využívajúcich rovnaké kompresné (kódovacie) princípy. Často je v literatúre uvedené, že vznikla nová kompresná metóda (ktorá však väčšinou len využíva známe kompresné princípy). Len veľmi zriedkavo vznikne nový kompresný princíp (využitie markovovských modelov, metóda LZMA). Kompresné princípy sa nezaoberajú implementačnými detailmi. Túto časť problému rieši až kompresná metóda, ktorá je veľmi často spojená s nejakým dátovým formátom napr.: .zip, .arj, .rar, a pod. [1].

1.4 Typy kompresných metód

Štatistické metódy kompresie - sú založené na pravdepodobnosti, s akou sa vyskytujú jednotlivé znaky v texte.

Slovníkové metódy kompresie - princípom týchto metód je vyhľadávanie opakujúcich sa častí textu. Do skomprimovaného textu sa uloží len prvý výskyt takej časti. Všetky ďalšie výskyty sú nahradené odkazom na predchádzajúci výskyt.

Kompresia nepohyblivého obrazu - táto časť je venovaná stratovej kompresii rastrového obrazu, ktorá poskytuje vysoký kompresný pomer.

Kompresia pohyblivého obrazu - v tejto časti ide o popis stratovej metódy kompresie s vysokým kompresným pomerom.

Kompresia zvuku - obsahuje popis princípov používaných pri stratovej kompresii zvuku[1].

1.5 Rozdelenie kompresných princípov

Základné delenie:

- Stratové - transformácia pomocou DCT (u JPEG), fraktálna kompresia, medzisnímková kompresia (u MPEG)
- Nestratové
  - Primárne - RLE, LZW, LZ77
  - Sekundárne - Huffmanova, Aritmetická kompresia

Delenie podľa výpočtovej náročnosti:

- Symetrické algoritmy – rovnaký čas potrebný pre kompresiu aj dekompresiu
- Asymetrické algoritmy – čas potrebný pre kompresiu a dekompresiu sa líši

Delenie podľa počtu priechodov:

- Jedno-priechodové
- Viac-priechodové

1.6 Metódy kompresie:

Jednoduché – založené na kódovaní opakujúcich sa postupností znakov (RLE)

Štatistické – založené na početnosti výskytu znakov v komprimovanom súbore (Huffmanovo kódovanie, aritmetické kódovanie)

Slovníkové – založené na kódovaní všetkých vyskytujúcich sa postupností

Transformačné – založené na ortogonálnych, popr. iných, transformáciách (JPEG, waveletová kompresia, fraktálová kompresia)

2. Slovníkové kompresné metódy

2.1 História

Slovníkové metódy využívajú fakt, že niektoré slová sa vo vstupnom reťazci vyskytujú častejšie. Tieto opakujúce sa slová ukladáme do slovníku. Vo výstupnom reťazci sú potom tieto slová nahradené im odpovedajúcimi kódovými slovami.

Slovníkové metody sú založené na vyhľadávaní opakujúcich sa častí textu. Pri kompresii sa do výstupného súboru uloží len prvý výskyt časti textu a dalšie výskyty sa nahradia odkazom. Podľa toho, ako tento odkaz rieši, delia sa slovníkové metódy na dve základné triedy:

1) Prvá trieda metód nahradzuje výskyt opakujúcej sa časti textu odkazom na miesto, kde se v texte táto časť predtým vyskytla.

2) Druhá trieda si z častí textu, ktoré sa už predtým vyskytli, vytvára v pamäti slovník a ďalšie výskyty nahradzuje odkazom na príslušné miesto v slovníku.

Slovníkovú kompresiu založenú na prvom uvedenom princípe publikovali páni Ziv a Lempel v roku 1977. Ich pôvodný návrh bol postupom času rôznymi spôsobmi modifikovaný a vznikla tak celá trieda kompresných metód, pre ktoré sa používa označenie LZ77. Rovnakí autori o rok neskôr publikovali ďalšiu kompresnú metódu, tentokrát založenú na druhom uvedenom princípe. Opäť z ich pôvodného návrhu postupom času vznikla celá trieda kompresných metód označovaná LZ78. Metódy obidvoch tried LZ77 a LZ78 sa zaraďujú do kategórie metód s adaptívnym modelom[1].

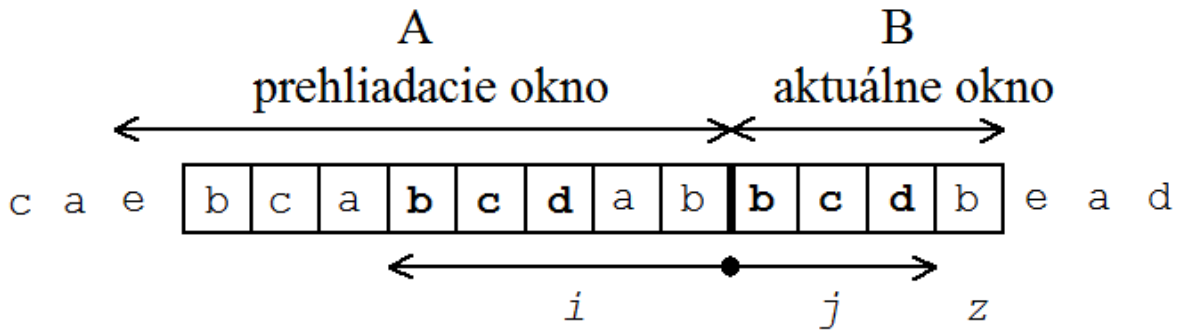
2.2 LZ77

Kódovanie LZ77 (nestrátové, primárne, jedno-priechodové, asymetrické). Pôvodná myšlienka pochádza z roku 1977 A. Lempel, J. Ziv. Metóda sa snaží nájsť vo vstupných dátach opakujúce sa sekvencie symbolov a tie na výstup predať vo forme odkazu na predchádzajúce (už odoslané) dáta. Odkazom je myslený údaj o pozícii opakujúcej sa sekvencie v predchádzajúcich dátach. Druhým dôležitým parametrom odkazu je dĺžka opakujúcej sa sekvencie.

V praxi je naviac nutné odlíšiť bežné dáta od odkazov, aby sme pri dekompresii boli schopní dáta správne dekódovať. Možností je niekoľko, bežne používanou je rozšírenie symbolov o ďalší bit, ktorý rozhodne, či sa jedná o dáta alebo odkaz (vhodné v prípade následného využitia niektorej z metód sekundárnej kompresie). Metóda je jedno-priechodová, čo je výhodná vlastnosť, najmä pri kompresii veľkého objemu dát na hardwarovej úrovni. Metóda je silno asymetrická, výpočtovú náročnosť pri kompresii je možné znížiť použitím binárneho vyhľadávacieho stromu.

Predstavme si, že máme akési prehliadacie okno, v ktorom vidíme časť už spracovanej časti vstupného reťazca s velikosťou A (typicky $A=2^{12}$). Ďalej máme aktuálné okno

s veľkosťou *B* (typicky $B=2^4$), v ktorom je práve kódované slovo. Aktuálne okno vo vstupnom reťazci bezprostredne nadväzuje na prehliadacie okno. Oknami pohybujeme a tým meníme obsah slovníka.



Obrázok č.1: Znázornenie metódy LZ77

Na začiatku vyplníme prehliadacie okno medzerami a v aktuálnom okne bude prvých *M* znakov vstupného reťazca. Vo vstupnom reťazci hľadáme dvojicu najdlhších, ale kratších ako *M*, zhodných podreťazcov *A* a *B* takých, že prvý znak reťazca *A* je v prehliadacom okne a *B* začína na prvom znaku v aktuálnom okienku. *B* je fráza, ktorú zakódujeme pomocou polohy *A* v prehliadacom okne. Kódové slovo frázy *B* je tvorené trojicou *(i, j, z)*, kde *i* je vzdialenosť (merané počtom znakov) prvého znaku reťazca *A* od začiatku aktuálneho okienka, *j* je dĺžka reťazca *B* a *z* je znak bezprostredne nasledujúci za reťazcom *B*. Po zapísaní kódového slova do výstupného reťazca sa obe okná posunú o *j+1* znakov doprava. Ak neexistuje reťazec *A* požadovaných vlastností, zapíšeme do výstupného reťazca trojicu *(0, 0, z)*, kde *z* je prvé písmeno aktuálneho okienka[1][2].

```
1:  begin
2:    naplň nezakódovanú časť okna zo vstupu
3:    while (nezakódovaná časť nie je prázdna) do
4:     begin
5:       nájdi v okne predponu p nezakódovanej časti začínajúcej v zakódovanej časti
6:       i := pozícia predpony p v okne
7:       j := dĺžka predpony p
8:       z := prvý znak za p v nezakódovanej časti
9:       output(i,j,z)
10:        pridaj do okna z prava i+1 znak
11:     end
12:  end
```

Výpis č.1: Pseudokód LZ77

2.3 LZSS

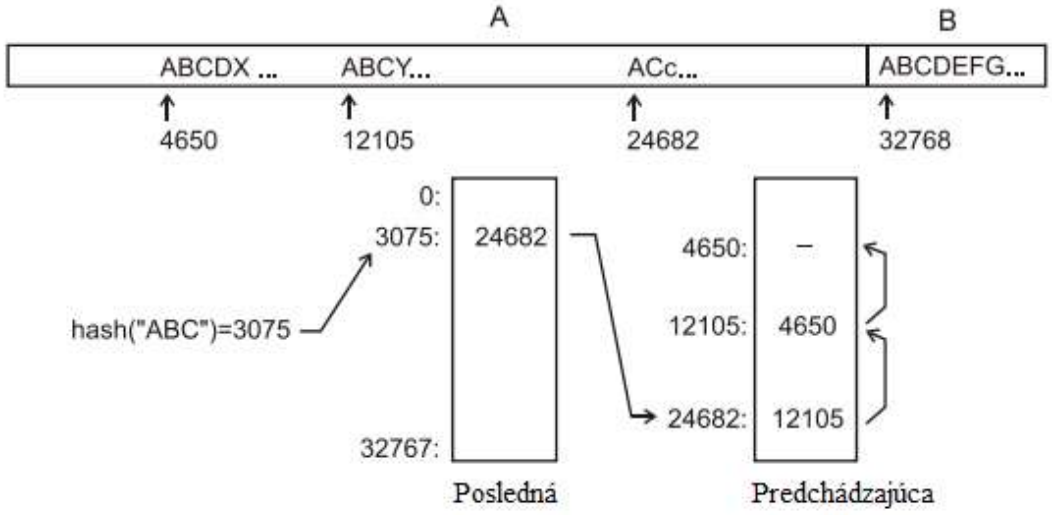
Lempel–Ziv–Storer–Szymanski – nestratový kompresný algoritmus, vytvorený v roku 1982**.** Kompresný algoritmus LZSS vychádza z algoritmu LZ77. Základom je opäť posuvné okno obsahujúce slovníkovú časť a čiastkový úsek nekomprimovaného textu. Rozdielne sú algoritmy hlavne v podobe výstupu, čo tiež predznamenáva niektoré zmeny v samotnom postupe kompresie. Výstup kompresnej metódy LZSS obsahuje dva typy dát. Ak sa nevyplatí zápis pomocou odkazu do slovníka, je na výstup zapisovaný pôvodný reťazec symbolu, ak sa však zápis s odkazom do slovníka vyplatí, zapíše sa ako dvojica odpovedajúca pozícii prvého symbolu v slovníku a počtu zhodných znakov. Na rozdiel od LZ77 sa už nezapisuje nezhodný znak[8].

O tom, či je výhodnejšie zapísať na výstup nekomprimovaný reťazec či odkaz do slovníka v podstate rozhoduje veľkosť posuvného okna. Veľkosti každej z oboch častí posuvného okna determinuje počet bitov na koľkých je možné zakódovať zhodu v slovníku. Napríklad, ak má slovníková časť veľkosť 20 znakov a časť pre načítané nekomprimované symboly má veľkosť 5, je možné pozíciu v slovníku popísať 5 bitmi a maximálnu zhodu 4 bitmi. Dokopy je teda potrebných 9 bitov k zakódovaniu jedného kompresného kroku. Ak súčasne môžeme predpokladať, že na zakódovanie jedného symbolu správy je potrebných 8 bitov, potom je zrejmé, že kódovať zo slovníka práve jeden symbol nie je výhodné. V takom prípade je výhodnějšie tento symbol kopírovať do komprimovaných dát bezo zmeny. K tomu, aby bolo možné dáta spätne dekomprimovať je ešte potrebné rozlíšiť, kedy bol zapísaný na výstup nekomprimovaný znak a kedy bol naopak použitý odkaz do slovníka. Na tento účel poslúži bitové pole, ktorým začína každý blok komprimovaných dát a udáva ako sú v tomto bloku rozložené nezmenené a odkazované dáta. Logika dekompresie je pre odkazované dáta úplne totožná s metódou LZ77.

Rýchlosť kompresie je závislá na rýchlosti vyhľadávania fráz. Preto kompresné programy majú túto časť často písanú v asembleri, ktorý z hľadiska rýchlosti umožňuje napísať efektívnejšie vyhľadávanie ako programovacie jazyky vyššej úrovne. Iný spôsob, ako zrýchliť vyhľadávanie, je použitie tabuľky existujúcich fráz.

Pre vytvorenie tabuľky fráz a rychlé vyhľadávanie v nej sa používa hashovacia funkcia. Hashovacia funkcia na základe textu frázy určí miesto v tabuľke, na ktorom bude uložená informácia udávajúca pozíciu tejto frázy v okne A. Pretože frázy sú všeobecne rôzne dlhé, hodnota hashovacej funkcie sa počíta len z počiatočnej časti frázy, ktorej dĺžka je rovná najmenšej používanej dĺžke frázy. Najmenšiu dĺžku frázy je účelné zvoliť v počte 3 znakov, pretože kódovanie frázy s dĺžkou 2 znakov nie je efektívne. Kódovanie ofsetu a dĺžky frázy by na výstupu zabralo miesto porovnateľné s kódovaním dvoch samostatných znakov. Ďalšou záležitosťou je voľba veľkosti hashovacej tabuľky. Všeobecne počet prvkov tabuľky volíme buď ako prvočíslo, alebo ako mocninu čísla 2. Prvočíslo prispieva k rovnomernému rozloženiu hodnôt v hashovacej tabuľke, pričom mocnina 2 je výhodná pre výpočet operácie modulo, ktorá sa používa v hashovacej funkcii, pretože je v tomto prípade možné realizovať bitovú operáciu logického súčinu (and). Použitie tejto druhej alternatívy býva veľmi časté.

Na príklade uvedenom v nasledujúcom obrázku sa hľadá fráza textu ABCDE..., ktorý je obsiahnutý v okne B. Hodnota hashovacej funkcie pre prvé tri znaky ABC je 3075. Na tomto mieste je v poli *Posledná* pozícia 24682 fráza s touto hodnotou hashovacej funkcie. Následne je nutné porovnať text v okne B s textom na pozícii 24682 v okne A. V tomto prípade zistíme, že ide o inú frázu, ktorá má rovnakú hodnotu hashovacej funkcie. V poli *Predchádzajúca* na pozícii 24682 nájdeme pozíciu 12105 ďalšej frázy s hodnotou hashovacej funkcie 3075. Porovnaním zistíme, že na tejto pozícii je v okne A fráza dĺžky 3 vzhľadom k textu v okne B. Fráza je síce nájdená, ale prehľadávanie reťazca fráz pokračuje, pretože môže existovať ešte dlhšia fráza. Na pozícii 12105 nájdeme v poli *Predchádzajúca* pozíciu 4605 ďalšieho kandidáta na frázu. Porovnanie textu na pozícii 4605 v okne A s textom v okne B ukazuje, že sme našli frázu dĺžky 4. Táto fráza je v poli *Predchádzajúca* poslednou frázou s hodnotou 3075 hashovacej funkcie, čím hľadanie frázy v tomto kompresnom kroku končí.

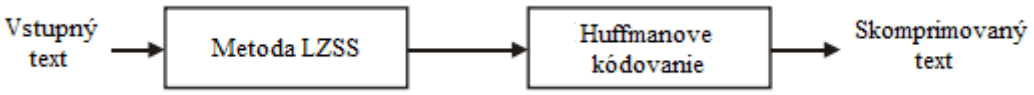


Obrázok č.2: Spôsob uloženia fráz v hashovacej tabuľke

Teraz prebehne zakódovanie nájdenej frázy. Ofset sa vypočíta ako rozdiel pozície okna B a pozície nájdenej frázy, tj. 32768-4650, dĺžka frázy je 4. Okno B sa posunie o 4 znaky doprava za nájdenou frázou, tj. na text EG... Zároveň s tým do tabuliek *Posledná* a *Predchádzajúca* zaradíme všetky frázy dĺžky 3, o ktoré bolo posunuté okno B. V našom príklade to budú 4 frázy ABC, BCD, CDE a DEG.

V tomto okamihu je obsah okna prekopírovaný na začiatok vyrovnávacej pamäte a prebehne aktualizácia tabuľky fráz, pri ktorej sú prepočítané pozície fráz obsiahnutých v presunutom okne a vylúčené frázy, ktoré sú mimo okno.

Výstupom metódy LZSS sú buď hodnoty ofsetu a dĺžky frázy, alebo hodnota znaku. Pre efektívne uloženie týchto hodnôt sa bežne používá Huffmanovo kódovanie. Typické usporiadanie kompresného programu je[1]:



Obrázok č.3: Základné usporiadanie kompresného programu

2.4 LZ78

Metóda (jedno-priechodová, adaptivná, symetrická) vymyslená v roku 1978 Abrahamom Lemplom a Jacobom Zivom. Z počiatku šlo o populárnu kompresnú metódu, ale po tom, čo boli časti algoritmu LZ78 patentované, záujem klesol.

Každý krok LZ78 pošle na výstup dvojicu (i,a), kde i je index frázy do slovníka a a je znak nasledujúci ihneď za nájdenou frázou. Slovník je prezentovaný ako strom s očíslovanými uzlami. Ak ideme od koreňa stromu do daného uzlu, dostaneme frázu zo vstupného textu[9][10].

V jednotlivých krokoch sa hľadá najdlhšia fráza v slovníku, ktorá by odpovedala zatiaľ nespracovanej časti vstupného textu. Index tejto fázy spolu s písmenom, ktoré nasleduje v texte za nájdenou časťou je potom poslané na výstup. Do slovníka je potom vložená stará fráza rozšírená o nové písmeno. Táto fráza je označená najmenším možným číslom.

Kódovanie začína so stromom čítajúcim jediný uzol, reprezentujúcim prázdny reťazec. Ide o pamäťovo náročnú metódu, pretože pre objemné vstupy môže slovník zabrať celú voľnú pamäť. Existuje niekoľko techník, vďaka ktorým je možné tomuto predísť. Pri nedostatku novej pamäte sa môže slovník zmraziť, zmazať a začať vytvárať nový, prípadne použiť aj iné metódy. V kompresnej utilite *compress* sa tento problém rieši zmrazením slovníka a následnou kontrolou kompresného pomeru. Ak tento prekročí povolenú hranicu, slovník sa zmaže a začne sa tvoriť nový.

Budeme komprimovať text *mamamaemuaemamamamu*.

		Nová fráza	
Vstupný text	Výstup	číslo	fráza
mamamaemuaemamamamu	(0,m)	1	m
amamaemuaemamamamu	(0,a)	2	a
mamaemuaemamamamu	(1,a)	3	ma
maemuaemamamamu	(3,e)	4	mae
muaemamamamu	(1,u)	5	mu
aemamamamu	(2,e)	6	ae
mamamamu	(3,m)	7	mam
amamu	(2,m)	8	am
amu	(8,u)	9	amu

Obrázok č.4: Znázornenie kompresie metódou LZ78

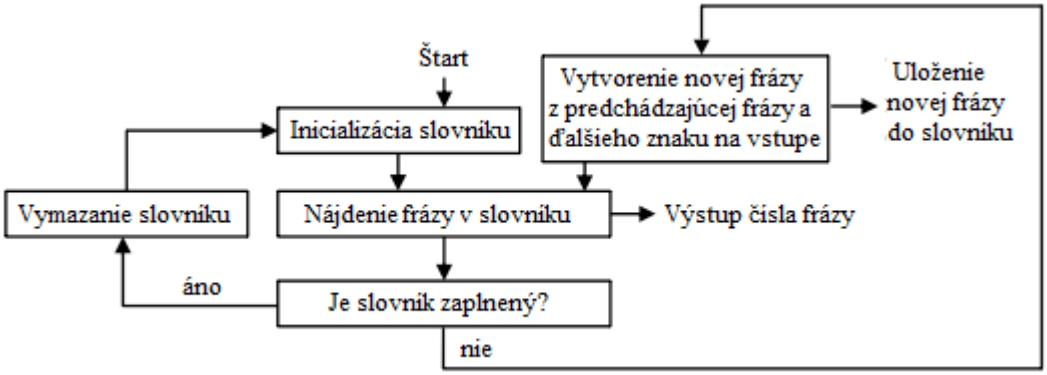
```
1:  begin
2:     inicializácia slovníku
3:     buffer := ""
4:     while (!koniecVstupu()) do
5:      begin
6:        symbol := ďalší nespracovaný znak zo vstupu
7:        if ( ( buffer.symbol ) je v slovníku ) then
8:           buffer := buffer.symbol;
9:        else
10:          begin
11:            output(indexDoSlovníku(buffer),symbol)
12:            pridaj do slovníku ( buffer.symbol )
13:            buffer := ""
14:          end
15:      end
16:     if ( !prázdny(buffer) ) then
17:        output(indexDoSlovníku(buffer));
18:  end
```

Výpis č.2: Pseudokód LZ78

2.5 LZW

LZW (Lempel-Ziv-Welch, nestratová, primárna, jedno-priechodová, asymetrická metóda). Metóda LZ78 bola modifikovaná v roku 1984 T. Welchem. Zo všetkých metód triedy LZ78, ktoré vznikli modifikovaním pôvodného prístupu popísaného v predchádzajúcej časti, má najväčší význam metóda LZW. Vytvorená metóda býva niekedy označovaná ako dictionary based encoding (slovníkovo orientované kódovanie). Metóda je patentovaná. Princíp vyhľadania predtým sa opakujúcich sekvencií ostala zachovaná, modifikovaný je spôsob kódovania. Metóda používa namiesto odkazov (pozícia, dĺžka) odkaz na položku slovníka[4]. Algoritmus postupne rozpoznáva a ukladá do tabuľky reťazce znakov a tieto reťazce nahradzuje vo výstupnom texte prirodzenými číslami z vopred definovaného intervalu. Tento interval určuje výslednú abecedu. Pretože metóda LZW nemôže kódovať samostatné znaky ale len nájdené frázy, nemôže byť slovník na začiatku kompresie prázdny. Je nutné ho inicializovať všetkými frázami dĺžky 1, tzn. je nutné do neho zaradiť všetky znaky, ktoré sa môžu vyskytnúť v texte. Napríklad pri kódovaní reťazca znakov zobrazených v osembitovom zobrazení je prvých 255 čísel vyhradených pre zobrazenie samostatných znakov z pôvodnej abecedy. Čísla nad 255 sa potom prideľujú jednotlivým nájdeným reťazcom. Pritom sa vytvára tabuľka (slovník) už rozpoznaných reťazcov.

Modifikáciou metódy je systematické premazávanie najdlhšie nepoužitých slov / položiek v slovníku namiesto zmazania celého slovníka. Metóda má v prípade varianty s pravidelným vymazávaním celého slovníka, menší kompresný pomer ako LZ77.



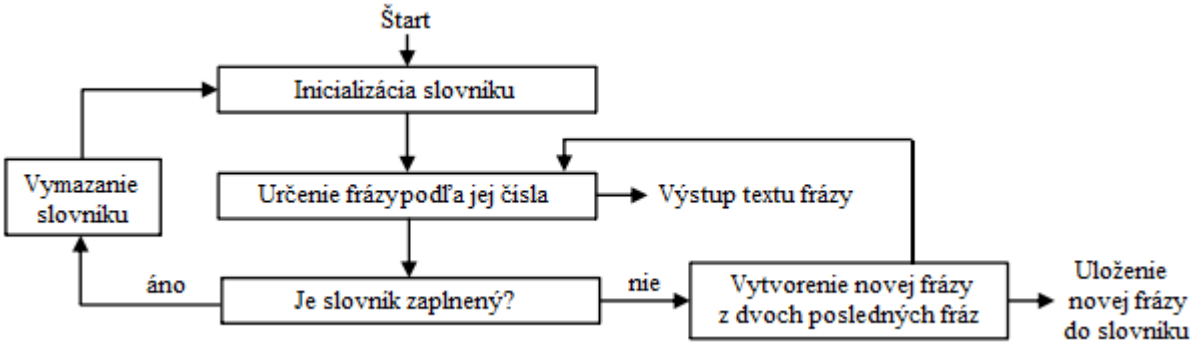
Obrázok č.5 : Schéma kompresie LZW

```
1:  begin
2:     while (not EOF(vstupný_súbor)) do
3:      begin
4:        readSymbol(znak)
5:        if ((reťazec+znak) in tabuľka_reťazcov) then
6:           reťazec += znak
7:        else
8:         begin
9:           output(číslo(reťazec, tabuľka_reťazcov))
10:            ADD(reťazec, tabuľka_reťazcov)
11:            reťazec := znak
12:         end
13:      end
14:  end
```

Výpis č.3: Pseudokód kompresie LZW

Popis dekompresie

Algoritmu dekompresie stačí len skomprimovaný reťazec, pretože je schopný zostaviť si z neho prekladovú tabuľku behom dekompresie. Avšak môže nastať prípad, kedy má reťazec na vstupe podobu znak/reťazec/znak/reťazec/znak (kde znak je nejaký rovnaký znak a reťazec je nejaký rovnaký reťazec) a v prekladovej tabuľke už máme uložený reťazec znak/reťazec. Akonáhle algoritmus dekompresie načíta znak/reťazec/znak, nemôže tento reťazec nájsť v tabuľke. Avšak tento stav je riešiteľný, pretože novo načítaný znak je zhodný s minule načítaným znakom.



Obrázok č.6: Schéma dekompresie LZW

```
1:  begin
2:     tmp = ""
3:     inicializácia slovníka
4:     while ((kód = načítaj kód) != EOF) do
5:      begin
6:        if ( existuje fráza v slovníku s kódom (kód) ) then
7:         begin
8:           fráza := slovník[kód]
9:           output(fráza)
10:            pridaj do slovníka (tmp + prvý znak z fráze)
11:         end
12:        else
13:         begin
14:           if ( kód > maximálny index slovníka + 1 )
15:              exit
16:           fráza := tmp + prvý znak z tmp
17:           output(fráza)
18:           pridaj do slovníka (fráza)
19:         end
20:        tmp := fráza
21:     end
22:  end
```

Výpis č.4: Pseudokód dekompresie LZW

2.6 Porovnanie kompresných metód

Niektoré kompresné metódy komprimujú lepšie binárne data, iné naopak textové súbory apod. Aby bolo možné porovnať vzájomnú činnosť rôznych kompresných metód a rôznych programov, bola zostavená referenčná vzorka obsahujúca 14 súborov s rôznymi typmi dát (anglické texty, obrázkové data, zdrojové texty programov, preložené programy). Tento súhrn sa nazýva *Calgary corpus* a účinnosť kompresie sa počíta ako priemer z kompresie všetkých 14 súborov.

V tabuľke sú uvedené metódy:

• Štatistická kompresia metódou PPMC (maximálna dĺžka kontextu 4, program Model-2).

• Slovníková kompresia LZ77 (metóda LZSS, program *gzip*).

• Slovníková kompresia LZ78 (metóda LZW, program *compress*).

• Kompresia blokovým triedením (testovací program realizujúci blokové triedenie, pričom pre uloženie je použité Huffmanove kódovanie). Doby kompresie a dekompresie sú v tabuľke uvedené v sekundách[1].

Tabuľka č.1: Porovnanie kompresných metód

Metóda	**Doba kompresie**	**Doba dekompresie**	**Kompresný pomer v %**
PPMC	603,2	614,1	30,4
LZ77-LZSS	42,6	4,9	33,9
LZ78-LZW	9,6	5,2	45,4
Blokové triedenie	51,1	9,4	30,9

3. Simulačné prostredie Octave

3.1 Základné vlastnosti a história

GNU Octave je vyšší programovací jazyk zameraný na numerické operácie. Je veľmi podobný jazyku MATLAB. Je vhodný pre tých, ktorým nerobí problémy algoritmizácia, ale vadia im zložité implementačné záležitosti, ktoré ich prenasledujú napr. pri programovaní v C++. Ocenia ho ale aj tí, ktorí potrebujú riešiť zložitejšie výpočty krok po kroku a nemajú k dispozícii vhodnú kalkulačku.

Autorom jazyka Octave je John W. Eaton a ako sa píše na domovskej stránke Octave aj mnoho dalších prispievateľov. Vznikol pôvodne ako pomôcka pre študentov, ktorým zaberalo príliš veľa času učiť sa Fortran, a mal im pomôcť s praktickými výpočtami. Octave je navrhnutý tak, aby tí, ktorí potrebujú niečo spočítať, nemuseli venovať veľa času učeniu sa nejakému „klasickému" programovaciemu jazyku. Posledná dokumentácia patrí k verzii 4.0.1. Celú dokumentáciu je možné nájsť na webových stránkach Octave.

Jazyk Octave je postavený na maticiach. Všetky operácie sú maticové, len niekedy sú tieto matice jednoprvkové. Matice môžu mať reálne aj imaginárne členy. Jazyk umožňuje aj numerickú integráciu sústav jednoduchých diferenciálnych rovníc alebo sústav jednoduchých diferenciálnych a algebraických rovníc.

Čas ušetrí okrem iného i fakt, že nie je potrebné premenné dopredu deklarovať a špecifikovať, či to bude typ celočíselný, reálny, alebo to budú písmená. Je možné vytvoriť aj dátovú štruktúru, ktorá bude obsahovať premenné všetkých typov a niektoré z nich budú jedno alebo dvojrozmerné (vektory alebo matice) – viď pôvodnú dokumentáciu. Ďalšou príjemnou vlastnosťou je to, že funkcia môže vracať akýkoľvek počet akýchkoľvek argumentov, pričom by sa programátor nemusel zaoberať ukazovateľmi.

Veľmi sympatickou vlastnosťou shellu je ukladanie postupnosti uskutočnených príkazov (= vzorcov) v súbore~/.octave_hist a ich možné vyhľadanie aj po reštarte shellu. Vyhľadávanie starších príkazov sa uskutočňuje šípkou hore. Ďalším spríjemnením práce je doplnenie príkazov na príkazovom riadku (pomocou tabulátoru), toto ale bohužiaľ neberie do úvahy históriu, ale iba používané premenné, príkazy Octave a názvy dostupných súborov (mj. skrípt a funkcií, ktoré sú v dosahu – tj. v aktuálnom adresári a v adresároch nastavených v premenných LOADPATH a DEFAULT_LOADPATH).

Ak chcete už dnes prácu ukončiť, ale úlohu ste ešte neukončili, nie je nič jednoduchšie, ako dôležité (alebo všetky) premenné uložiť. K tomu slúži príkaz *save file prom*, kde file je názov súboru, ktorý chcete pre uloženie použiť a prom je meno vašej premennej, ktorú chcete uložiť (môže byť aj viacej premenných, bez parametra uloží všetky vaše premenné). Keď si budete chcieť nabudúce premenné načítať, stačí zadať príkaz *load file* (kde file je opäť meno súboru). Oba tieto príkazy majú ja prepínače (určujú formát súboru) – bližšie viď. pôvodná dokumentácia. Pre úplnosť dodajme ešte dva príkazy pre prácu s priestorom premenných (tzv. workspace) a to *who* (zobrazí všetky vami používané premenné) a *clear* (všetky premenné vymaže).

V shelle samozrejme fungujú základné operácie pre prácu so systémom súborov – operácie cd, ls apod., pokiaľ nie sú prepísané premennou.

Interpreter Octave (so shellom) sa dá stiahnuť na domovskej stránke Octave a je súčasťou linuxových distribúcií Debian, SuSe a RedHat. Distribuuje sa pod GNU GPL licenciou[5].

3.2 Verzia a inštalácia

Pre simulácie som si zvolil GNU Octave verzie 4.0.0 ktorá je voľne dostupná od mája 2015 na stránkach GNU a obsahuje nové grafické uživateľské rozhranie a má väčšiu kompatibilitu s nástrojom Matlab. Od marca 2016 je dostupná už aj verzia 4.0.1., kde bolo opravených viacero chýb. Program som používal pod OS Windows 10 Education x64.

Po stiahnutí programu nainštalujeme jednoduchým spustením exe súboru. V OS GNU/Linux založenom na Debiane môže správca Octave nainštalovať príkazom *apt-get install octave* . Program sa spúšťa príkazom *octave* .

3.3 Práca s balíčkami

Pri simuláciach využívam nasledovné balíčky:
Balíčky si zobrazíme príkazom *pkg list*

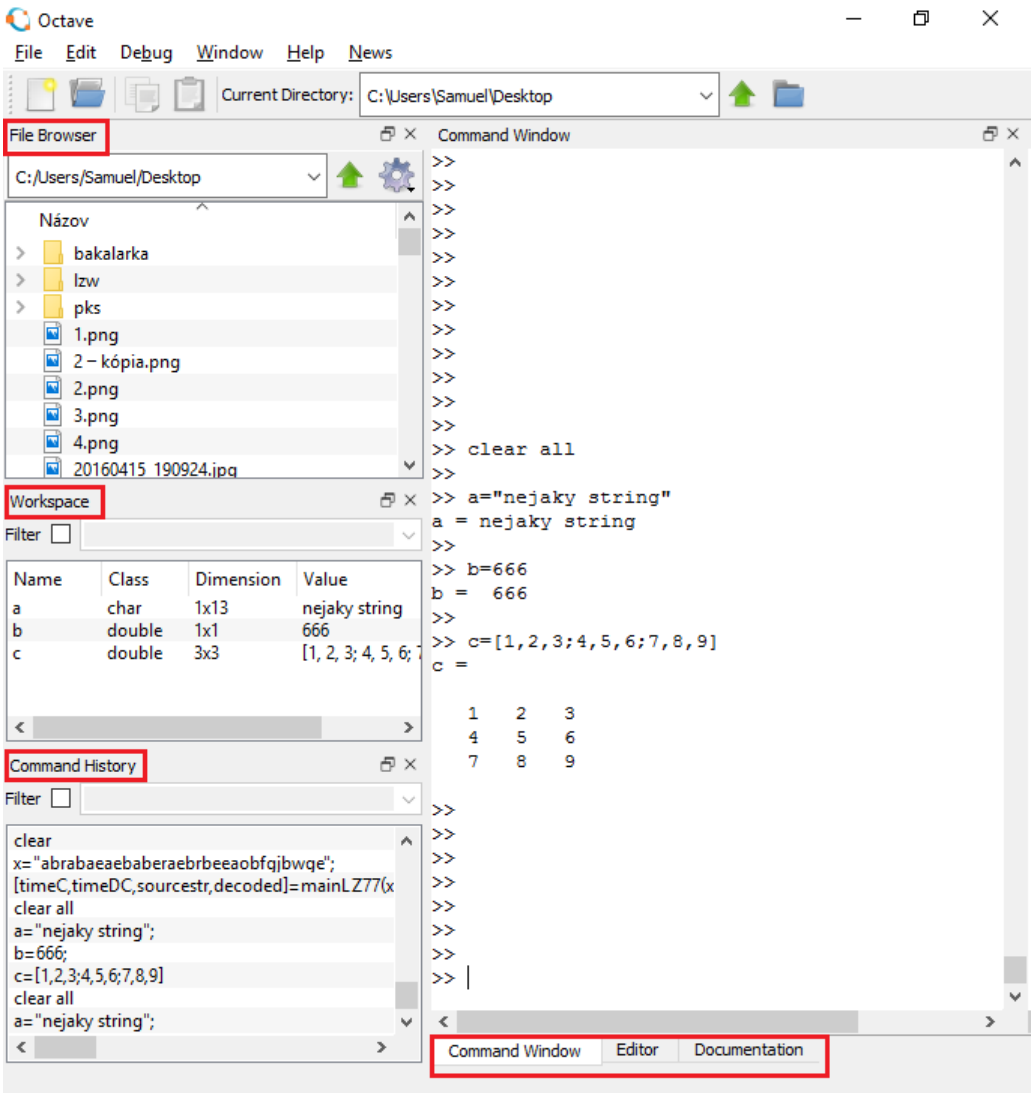
```
>>
>> pkg list
Package Name    | Version | Installation directory
----------------+---------+-----------------------
communications  |   1.2.1 | C:\Octave\Octave-4.0.0\share\octave\packages\communications-1.2.1
       control  |   3.0.0 | C:\Octave\Octave-4.0.0\share\octave\packages\control-3.0.0
 dvbldpc-iscml  |   1.0.1 | C:\Octave\Octave-4.0.0\share\octave\packages\dvbldpc-iscml-1.0.1
        signal  |   1.3.2 | C:\Octave\Octave-4.0.0\share\octave\packages\signal-1.3.2
>>
```

Výpis č.5: Octave – nainštalované balíčky

Balíčky stiahneme na stránke [6]. Balíček nainštalujeme príkazom *pkg install názov_balíčku*, napr. *pkg install communications-1.2.1.tar.gz* , následne použijeme príkaz *pkg update* pre aktualizovanie a príkaz *pkg load all* pre nahratie balíčkov. Pre odobratie, alebo odinštalovanie používame príkazy *pkg unload názov_balíčka* a *pkg uninstall názov_balíčka* . Zoznam všetkých príkazov pre prácu s balíčkami nájdeme na stránke[15].

3.4 Popis prostredia Octave (GUI)

K simuláciam nepoužívam Octave (CLI), ale Octave (GUI), ktoré by malo v budúcnosti plniť funkciu základného rozhrania.



Obrázok č.7: GUI Octave

File Browser – slúži na prehľadávanie súborov, ktoré tu môžme priamo otvoriť

Workspace – prehľad zadeklarovaných premenných a výstupov funkcií

Command History – história zadaných príkazov

Command Window – príkazový riadok

Editor – slúži na tvorbu funkcií, skriptov a úpravu súborov

Documentation – užívateľský manuál, ktorý obsahuje aj zoznam nových možností aktuálnej verzie

3.5 Práca so skriptami / funkciami / m-súborom

M-súbor je textový súbor, do ktorého vkladáme Octave príkazy. Keď súbor spustíme, príkazy sú prečítané a spustené presne ako keby sme ich postupne zadali do príkazového riadku Octave. Všetky m-súbory musia končiť príponou .m (napr. *skript.m*).

Ich využívanie nám uľahčuje a zrýchľuje prácu s príkazmi. Ak potrebujeme otestovať viacero parametrov zviazaných s väčším počtom príkazov, snažíme sa využívať skripty.

Skripty slúžia k preneseniu príkazov z príkazového riadku do súboru. To súvisí so spôsobom zachádzania s premennými. Skript môže pristupovať k už všetkým zavedeným premenným, nielen k tým, ktoré sú vytvorené priamo v skripte, ale aj k tým, ktoré si užívateľ vytvoril v príkazovom riadku ešte pred spustením skpriptu. Všetky premenné, ktoré boli vytvorené v skripte, sú naďalej prístupné aj po jeho skončení. Ak bol obsah premenných v skripte zmenený, ostane zachovaný aj po ukončení skriptu.

Funkcia sa od skriptu líši tým, že musí mať na začiatku uvedené kľúčové slovo *function ,*
za ktorým nasleduje názov funkcie. Funkcia môže pracovať len s premennými, ktoré sú zavedené v jej tele, alebo deklarované ako vstupné či výstupné premenné. Žiadne iné premenné nie sú vo funkcii viditeľné, takže ich hodnoty a obsah nie je možné načítať.

Vytvorenie, ukladanie a otváranie
PC alebo MAC:

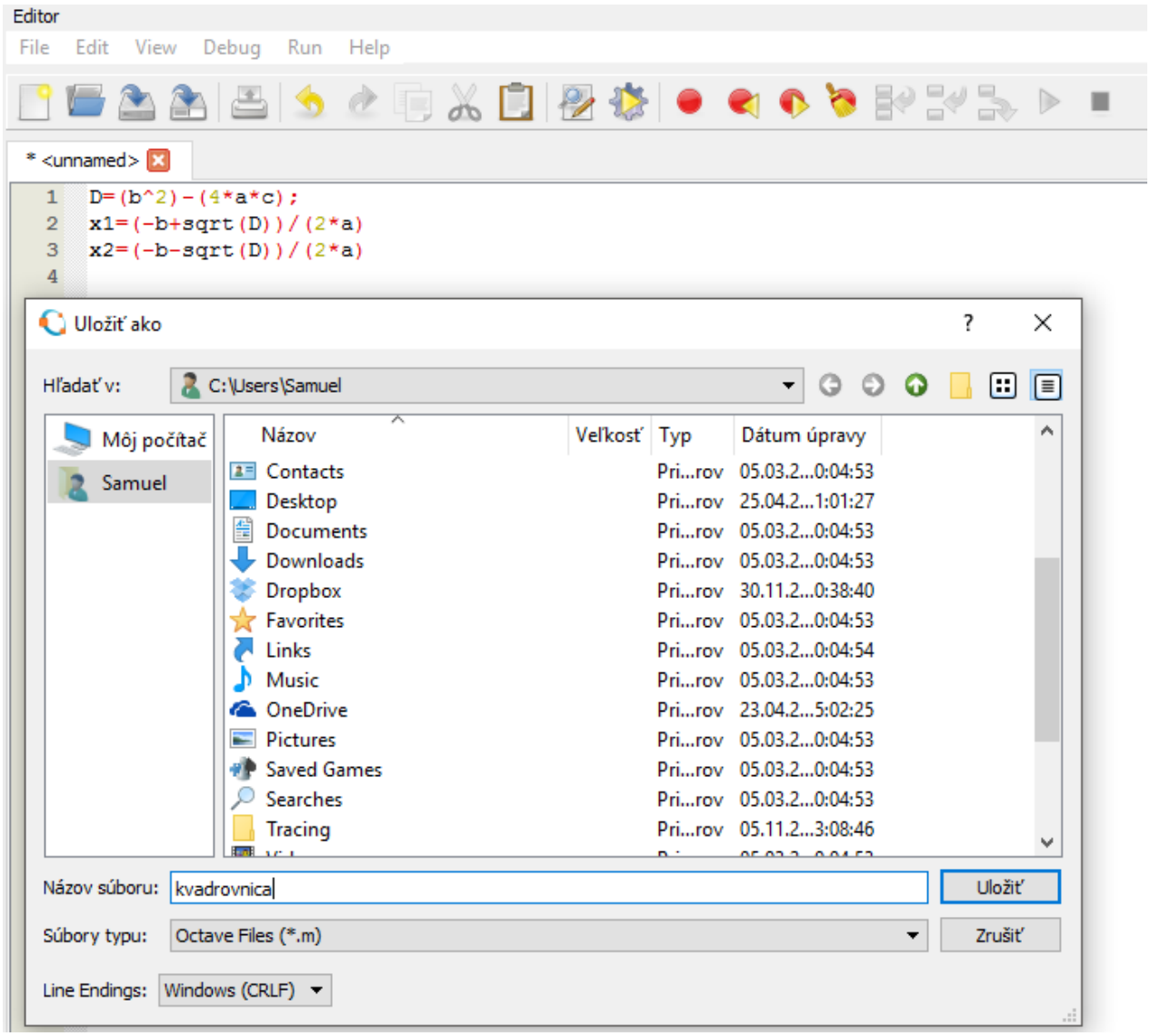
Pre vytvorenie klikneme na položku *File -> New* a vyberieme položku *New Script / New Function*. Pri funkcii zadávame aj jej názov a v novom *Editor Window* môžeme pridávať nové Octave príkazy.

Ak chceme uložiť m-súbor, jednoducho prejdeme do *File* menu a vyberieme *Save File / Save File as...* Súbor musíme uložiť s príponou .m .

Pre otvorenie existujúceho m-súboru vyberieme položku *File -> Open* .

3.5.1 Tvorba skriptu

Vytvoríme si primitívny skript vypočítania koreňov kvadratickej rovnice, ktorý uložíme pod názvom *kvadrovnica.m* .



Obrázok č.8: Vytvorenie a uloženie skriptu

.

```
>> a=3;
>> b=6;
>> c=-9;
>>
>> kvadrovnica
x1 =  1
x2 = -3
```

Výpis č.6: Zadanie parametrov a výstup skriptu

UNIX:

Pre vytvorenie m-súboru použijeme niektorý z textových editorov (napr. pico, nedit, vi, emacs atď.), súbor musí obsahovať príponu .m .

Spustenie

Po tom čo je m-súbor vytvorený, obsahuje príponu .m a nachádza sa v Octave adresári (alebo sa k nemu dostaneme prostredníctvom *File Browseru*), môžeme príkazy v m-súbore spustiť jednoducho zavolaním názvu m-súboru v Octave príkazovom riadku.

V prípade, že náš m-súbor nemá vopred zadefinované všetky potrebné premenné a nechceme ich zadávať priamo pri volaní danej funkcie, môžme si m-súbor jednoducho otvoriť z akéhokoľvek miesta dvojklikom.

Ak nechceme spustiť všetky príkazy ktoré m-súbor obsahuje, môžeme skopírovať len jeho časť, ktorú potrebujeme a vložiť do príkazového riadku[14].

4. Simulácia kompresie a dekompresie

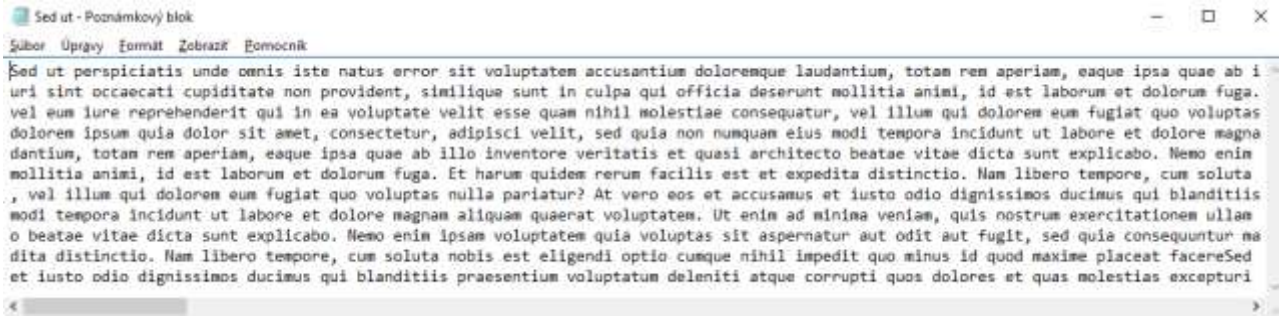
4.1 Simulácia kompresie a dekompresie metódou LZ77

Ukážka kompresie a dekompresie jednoduchého stringu *x*. Po zadeklarovaní stringu *x* a zavolaní funkcie *mainLZ77* môžme vidieť výpis kompresie, potrebný čas pre kompresiu *timeC* a dekompresiu *timeDC* a výpis zadaného stringu *sourcestr* a spätne zloženého stringu *decoded*.

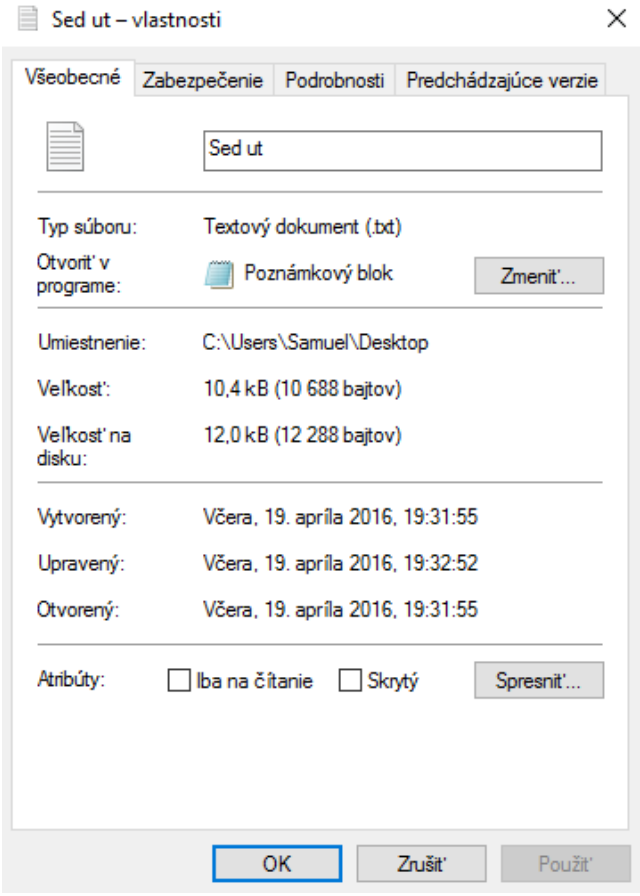
```
>> x="simulacia kompresie a dekompresie simulacia kompresie a dekompresie
simulacia kompresie a dekompresie";
>>
>> [timeC,timeDC,sourcestr,decoded]=mainLZ77(x)
LZ77-Compression is started.
<0,0,C(s)>
<0,0,C(i)>
<0,0,C(m)>
<0,0,C(u)>
<0,0,C(l)>
<0,0,C(a)>
<0,0,C(c)>
<6,1,C(a)>
<0,0,C( )>
<0,0,C(k)>
<0,0,C(o)>
<10,1,C(p)>
<0,0,C(r)>
<0,0,C(e)>
<16,2,C(e)>
<10,1,C(a)>
<2,1,C(d)>
<5,1,C(k)>
<14,9,C(s)>
<34,34,C(i)>
<34,31,C($)>
 LZ77-Compression is finished.
 LZ77-Decompression is started.
 LZ77-Decompression is finished.ok =  1
timeC =  0.096064
timeDC =  0.075049
sourcestr = simulacia kompresie a dekompresie simulacia kompresie a
dekompresie simulacia kompresie a dekompresie$
decoded = simulacia kompresie a dekompresie simulacia kompresie a dekompresie
simulacia kompresie a dekompresie$
```

Výpis č.7: Kompresia a dekompresia stringu x

Ďalšiu simuláciu som sa rozhodol uskutočniť na väčšom objeme dát. Vytvoril som textový súbor *Ted ut* (10688 bajtov), do ktorého som z generátora textu viac krát nakopíroval rovnaký vygenerovaný text. Súbor obsahoval 10688 znakov.

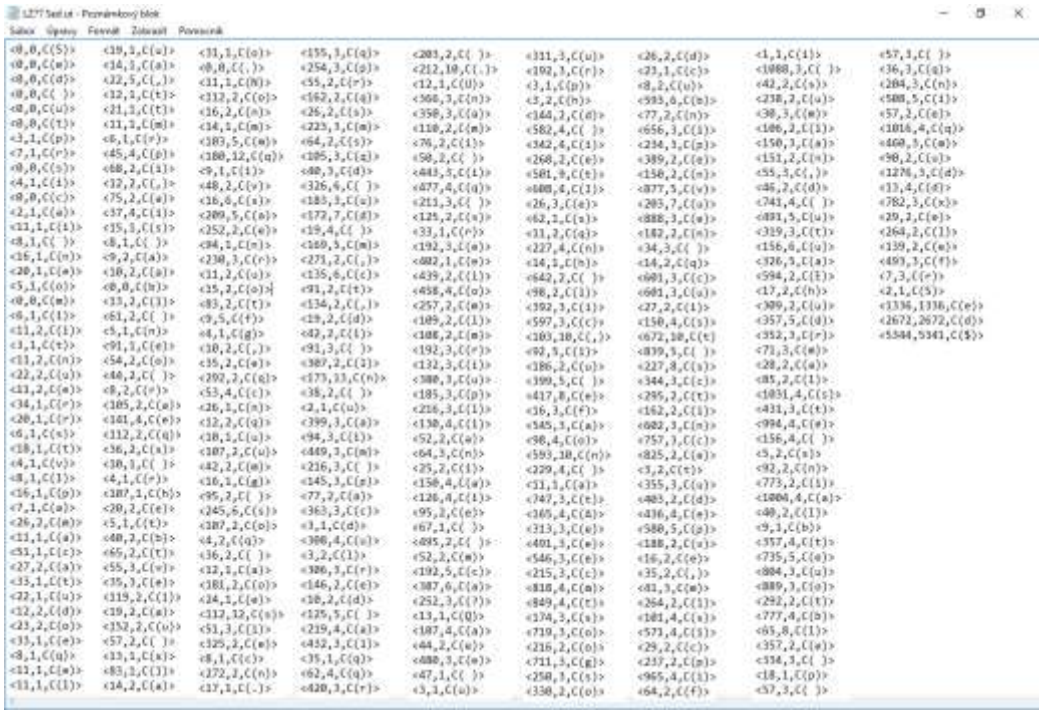


Obrázok č.9: Textový súbor Ted ut

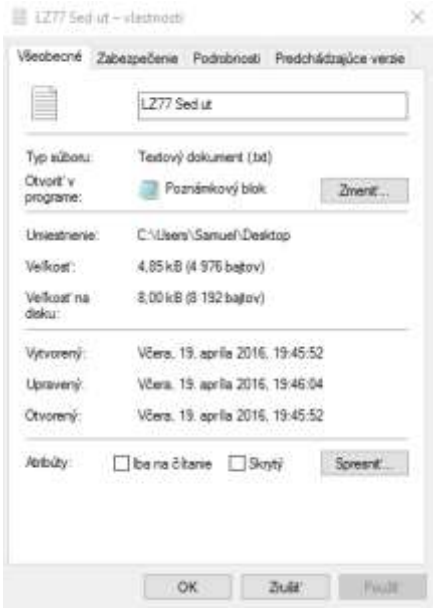


Obrázok č.10: Textový súbor Ted ut – vlastnosti

Skomprimovaný textový súbor *Ted ut* (4976 bajtov). Ten sa mi kompresným algoritmom podarilo zmenšiť na 46,5% pôvodnej veľkosti. Tento výsledok pripisujem faktu, že sa v súbore opakoval rovnaký text a pri dostatočne veľkej nastavenej veľkosti okien som dosiahl tak dobrý kompresný pomer. Čas potrebný na kompresiu bol 14,8 sekundy. Čas potrebný na dekmpresiu 14,4 sekundy.



Obrázok č.11: Výpis kompresie súboru Ted ut



Obrázok č.12: Skomprimovaný súbor Ted ut – vlastnosti

4.1.1 Postup simulácie v GNU Octave:

Otvoríme si m-súbor s názvom *mainLZ77*. Vo funkcii si v prvom rade upravíme dĺžku prehliadacieho a aktuálneho okna. Tieto parametre si volíme podľa množstva textu a predpokladaného opakovania textu. Aby sme mohli pracovať so zmenenými parametrami, m-súbor uložíme.

```
searchWindowLen=
lookAheadWindowLen=
```

Zadeklarujeme si string, na ktorom chceme aby prebehla kompresia / dekompresia.

```
sourcestr=” sem vlozime text ”;
```

Na zadeklarovaný string si zavoláme hlavnú funkciu z m-súboru.

```
[timeC,timeDC,sourcestr,decoded]=mainLZ77(sourcestr)
```

Dostávame potvrdenie o začatí kompresie, po ktorom prebehne výpis skomprimovaného stringu.

```
LZ77-Compression is started.
```

Po úspešnom ukončení kompresie dostávame potvrdnie o ukončení kompresie a hlásenie o začatí dekompresie. Opäť nám funkcia potvrdzuje ukončenie dekompresie a vypisuje zhodu dekomprimovaného stringu.

```
LZ77-Compresssion is finished.
LZ77-Decompression is started.
LZ77-Decompression is finished.ok =  1
```

Následuje výpis času potrebného na kompresiu a dekompresiu.

```
timeC =

timeDC =
```

Ďalší výpis obsahuje originálny string, ktorý sme deklarovali na začiatku.

```
sourcestr=
```

Nakoniec dochádza k spätnému zloženiu skomprimovaného stringu na string zadaný – dekompresia.

```
decoded=
```

4.1.2 Ukážka kódu funkcie pre dekompresiu:

```
function decompressed=decode(binaryStr,searchWindowLen,
lookAheadWindowLen)

decompressed='';
bytenumSW=length(dec2bin(searchWindowLen));
bytenumLA=length(dec2bin(lookAheadWindowLen));
i=1;

while i<length(binaryStr)
      SW=returnPartOfString(binaryStr,i,i-1+bytenumSW);
      SWdec=bin2dec(SW);
      i=i+bytenumSW;
      if(SWdec~=0)
            LA=returnPartOfString(binaryStr,i,i-1+bytenumLA);
            LAdec=bin2dec(LA);
            i=i+bytenumLA;
      else
            LAdec=0;
      end

      Chr=returnPartOfString(binaryStr,i,i-1+8);
      Chrch=char(bin2dec(Chr));
      i=i+8;

      if(SWdec==0)
            decompressed=strcatNew(decompressed,Chrch);

      else
            location=length(decompressed)-SWdec;

            for j=1:LAdec
            decompressed=strcatNew(decompressed,decompressed
            (location+j));

            end
            decompressed=strcatNew(decompressed,Chrch);

      end
end
end
```

Výpis č.9: Funkcia dekompresie LZ77

4.2 Simulácia kompresie a dekompresie metódou LZW

Ukážka kompresie a dekompresie stringu *str*, ktorý sme si po otvorení m-súboru s názvom *lzw_demo1* zadeklarovali. Pri deklarácii sme použili funkciu uint8(x), ktorá nám string previedla na tvar unsigned 8-bit integer.

```
>> str = uint8("mamamaemuaemamamamumamamaemuaemamamamumamamaemuaemamamamu");
```

Výpis č.10: Deklarácia stringu v tvare unsigned 8-bit integer

Po volaní funkcie *norm2lzw* na zadeklarovaný string dostávame na výstupe výpis kompresie (*packed*=) a výpis slovníku (*table*=), ktorý pozostáva z ASCII tabuľky (255 znakov) a

```
>> [packed,table]=norm2lzw(uint8(str))

packed = 109   97  256  256  101  109  117   97  260  257  265  261  258  265
260  262  264  268  267  266  263  261  276  275  261

table =
{
  [1,1] = 0     [1,40] = 39   [1,80] = 79    [1,120] = 119   [1,160] = 159   [1,200] = 199   [1,240] = 239
  [1,2] = 1     [1,41] = 40   [1,81] = 80    [1,121] = 120   [1,161] = 160   [1,201] = 200   [1,241] = 240
  [1,3] = 2     [1,42] = 41   [1,82] = 81    [1,122] = 121   [1,162] = 161   [1,202] = 201   [1,242] = 241
  [1,4] = 3     [1,43] = 42   [1,83] = 82    [1,123] = 122   [1,163] = 162   [1,203] = 202   [1,243] = 242
  [1,5] = 4     [1,44] = 43   [1,84] = 83    [1,124] = 123   [1,164] = 163   [1,204] = 203   [1,244] = 243
  [1,6] = 5     [1,45] = 44   [1,85] = 84    [1,125] = 124   [1,165] = 164   [1,205] = 204   [1,245] = 244
  [1,7] = 6     [1,46] = 45   [1,86] = 85    [1,126] = 125   [1,166] = 165   [1,206] = 205   [1,246] = 245
  [1,8] = 7     [1,47] = 46   [1,87] = 86    [1,127] = 126   [1,167] = 166   [1,207] = 206   [1,247] = 246
  [1,9] = 8     [1,48] = 47   [1,88] = 87    [1,128] = 127   [1,168] = 167   [1,208] = 207   [1,248] = 247
  [1,10] = 9    [1,49] = 48   [1,89] = 88    [1,129] = 128   [1,169] = 168   [1,209] = 208   [1,249] = 248
  [1,11] = 10   [1,50] = 49   [1,90] = 89    [1,130] = 129   [1,170] = 169   [1,210] = 209   [1,250] = 249
  [1,12] = 11   [1,51] = 50   [1,91] = 90    [1,131] = 130   [1,171] = 170   [1,211] = 210   [1,251] = 250
  [1,13] = 12   [1,52] = 51   [1,92] = 91    [1,132] = 131   [1,172] = 171   [1,212] = 211   [1,252] = 251
  [1,14] = 13   [1,53] = 52   [1,93] = 92    [1,133] = 132   [1,173] = 172   [1,213] = 212   [1,253] = 252
  [1,15] = 14   [1,54] = 53   [1,94] = 93    [1,134] = 133   [1,174] = 173   [1,214] = 213   [1,254] = 253
  [1,16] = 15   [1,55] = 54   [1,95] = 94    [1,135] = 134   [1,175] = 174   [1,215] = 214   [1,255] = 254
  [1,17] = 16   [1,56] = 55   [1,96] = 95    [1,136] = 135   [1,176] = 175   [1,216] = 215   [1,256] = 255
  [1,18] = 17   [1,57] = 56   [1,97] = 96    [1,137] = 136   [1,177] = 176   [1,217] = 216
  [1,19] = 18   [1,58] = 57   [1,98] = 97    [1,138] = 137   [1,178] = 177   [1,218] = 217
  [1,20] = 19   [1,59] = 58   [1,99] = 98    [1,139] = 138   [1,179] = 178   [1,219] = 218
  [1,21] = 20   [1,60] = 59   [1,100] = 99   [1,140] = 139   [1,180] = 179   [1,220] = 219
  [1,22] = 21   [1,61] = 60   [1,101] = 100  [1,141] = 140   [1,181] = 180   [1,221] = 220
  [1,23] = 22   [1,62] = 61   [1,102] = 101  [1,142] = 141   [1,182] = 181   [1,222] = 221
  [1,24] = 23   [1,63] = 62   [1,103] = 102  [1,143] = 142   [1,183] = 182   [1,223] = 222
  [1,25] = 24   [1,64] = 63   [1,104] = 103  [1,144] = 143   [1,184] = 183   [1,224] = 223
  [1,26] = 25   [1,65] = 64   [1,105] = 104  [1,145] = 144   [1,185] = 184   [1,225] = 224
  [1,27] = 26   [1,66] = 65   [1,106] = 105  [1,146] = 145   [1,186] = 185   [1,226] = 225
  [1,28] = 27   [1,67] = 66   [1,107] = 106  [1,147] = 146   [1,187] = 186   [1,227] = 226
  [1,29] = 28   [1,68] = 67   [1,108] = 107  [1,148] = 147   [1,188] = 187   [1,228] = 227
  [1,30] = 29   [1,69] = 68   [1,109] = 108  [1,149] = 148   [1,189] = 188   [1,229] = 228
  [1,31] = 30   [1,70] = 69   [1,110] = 109  [1,150] = 149   [1,190] = 189   [1,230] = 229
  [1,32] = 31   [1,71] = 70   [1,111] = 110  [1,151] = 150   [1,191] = 190   [1,231] = 230
  [1,33] = 32   [1,72] = 71   [1,112] = 111  [1,152] = 151   [1,192] = 191   [1,232] = 231
  [1,34] = 33   [1,73] = 72   [1,113] = 112  [1,153] = 152   [1,193] = 192   [1,233] = 232
  [1,35] = 34   [1,74] = 73   [1,114] = 113  [1,154] = 153   [1,194] = 193   [1,234] = 233
  [1,36] = 35   [1,75] = 74   [1,115] = 114  [1,155] = 154   [1,195] = 194   [1,235] = 234
  [1,37] = 36   [1,76] = 75   [1,116] = 115  [1,156] = 155   [1,196] = 195   [1,236] = 235
  [1,38] = 37   [1,77] = 76   [1,117] = 116  [1,157] = 156   [1,197] = 196   [1,237] = 236
  [1,39] = 38   [1,78] = 77   [1,118] = 117  [1,158] = 157   [1,198] = 197   [1,238] = 237
                [1,79] = 78   [1,119] = 118  [1,159] = 158   [1,199] = 198   [1,239] = 238
```

z nových, rozpoznaných reťazcov.

```
[1,257] =
  109   97
[1,258] =
   97  109
[1,259] =
  109   97  109
[1,260] =
  109   97  101
[1,261] =
  101  109
[1,262] =
  109  117
[1,263] =
  117   97
[1,264] =
   97  101
[1,265] =
  101  109   97
[1,266] =
   97  109   97
[1,267] =
   97  109   97  109
[1,268] =
  109  117  109
[1,269] =
  109   97  109   97
[1,270] =
   97  109   97  101
[1,271] =
  101  109  117
[1,272] =
  117   97  101
[1,273] =
  101  109   97  109
[1,274] =
  109   97  109   97  109
[1,275] =
  109  117  109   97
[1,276] =
   97  109   97  109   97
[1,277] =
   97  101  109
[1,278] =
  109  117   97
[1,279] =
   97  101  109   97
[1,280] =
   97  109   97  109   97  109
}
```

Výpis č.11: Funkcia norm2lzw – výpis kompresie a slovníku

Následným volaním funkcie *lz2norm* sa vykoná dekompresia (*unpacked*=), ktorú vidíme na výstupe aj s výpisom slovníka (*table*=).

```
>> [unpacked,table]=lzw2norm(packed)
unpacked =

 Columns 1 through 28:

  109   97  109   97  109   97  101  109  117   97  101  109   97  109   97
109   97  109  117  109   97  109   97  109   97  101  109  117

 Columns 29 through 56:

   97  101  109   97  109   97  109   97  109  117  109   97  109   97  109
97  101  109  117   97  101  109   97  109   97  109   97  109

 Column 57:

  117

table = {}
```

Výpis č.12: Funkcia lzw2norm – výpis dekompresie

Výstup dekompresie(*unpacked*=) si prevedieme funkciou *char* na znaky(string). Môžme vidieť, že string sa zhoduje so stringom pôvodne zadaným.

```
>> unpacked = char(unpacked)
unpacked =
mamamaemuaemamamamumamamaemuaemamamamumamamaemuaemamamamu
```

Výpis č.13: Prevod výstupu dekompresie na znaky

Funkciou *strvcat(table{257:end})* docielime výpis všetkých nových reťazcov, ktoré boli pridané do slovníka.

```
>> strvcat(table{257:end})
ans =
ma
am
mam
mae
em
mu
ua
ae
ema
ama
amam
mum
mama
amae
emu
uae
emam
mamam
muma
amama
aem
mua
aema
amamam
```

Výpis č.14: Nové reťazce v slovníku

4.2.1 Ukážka kódu pre kompresiu

```
      output = vector;

outputindex = 1;
startindex = 1;
for index=2:length(vector),
      element = vector(index);
      substr = vector(startindex:(index-1));
      code = getcodefor([substr element],table);
      if isempty(code),
            output(outputindex) = getcodefor(substr,table);
            [table,code] = addcode(table,[substr element]);
            outputindex = outputindex+1;
            startindex = index;
      else,
      end
end
```

Výpis č.15: Funkcia kompresie LZW

4.2.2 Ukážka kódu pre dekompresiu

```
output = uint8([]);

code = vector(1);
output(end+1) = code;
character = code;
for index=2:length(vector),
      element = vector(index);
      if (double(element)+1)>length(table),
            string = table{double(code)+1};
            string = [string character];
      else,
            string = table{double(element)+1};
      end
      output = [output string];
      character = string(1);
      [table,code] = addcode(table,[table{double(code)+1}
character]);
      code = element;
end
```

Výpis č.16: Funkcia dekompresie LZW

Záver

Cieľom bakalárskej práce bolo štúdium a popis slovníkových kompresných metód, štúdium a popis simulačného prostredia Octave a vykonanie simulácií kompresie a dekompresie dát pomocou Octave.

V prvej, teoretickej časti tejto práce sa venujem všeobecnému uvedeniu do problematiky kompresie, jej históriou a využitím, zoznámeniu sa so základnými pojmami, základnými deleniami kompresie a kompresných princípov.

V druhej časti detailne rozoberám slovníkové kompresné metódy, ich históriu a zameriavam sa na štyri z najznámejších slovníkových nestratových algoritmov na kompresiu dát. Detailne popisujem základné metódy LZ77, LZ78 a metódy LZSS a LZW, ktoré sú z nich odvodené. Zaoberám sa ich históriou, spôsobom, ktorým metódy fungujú a podmienkami, pri ktorých je ich použitie výhodné. Pripájam príklady, schémy funkčnosti a pseudokódy algoritmov. Na záver tejto časti porovnávam kompresné pomery týchto metód s metódou PPMC a blokovým triedením vzhľadom na ich priemer účinnosti kompresie referenčnej vzorky súboru Calgary corpus.

V tretej časti sa zaoberám simulačným prostredím GNU Octave, jeho históriou a vlastnosťami. Stručne popisujem inštaláciu programu a potrebných balíčkov. Zvolil som si verziu 4.0.0 pod operačným systémom Windows 10 Education x64 a pracoval som s grafickým užívateľským rozhraním, ktoré by malo v budúcnosti plniť úlohu základného rozhrania. Pridávam základný popis rozhrania, opis práce a tvorby skriptov, funkcií a m-súborov.

V štvrtej, poslednej časti sa zaoberám realizáciou simulácii kompresie a dekompresie. Pre simulácie som si zvolil metódu LZ77, ktorá je zo zpomínaných metód najstaršia a metódu LZW, ktorá sa stala prvou univerzálnou metódou dátovej kompresie v počítačoch. Ako príklad pri LZ77 uvádzam najskôr kompresiu a dekompresiu jednoduchého stringu, pridávam výstup z Octave, ktorý zahŕňa výpis kompresie, čas potrebný na kompresiu, čas potrebný na dekompresiu a zloženie skomprimovaných dát na pôvodný string – dekomprimácia. Ďalej pracujem s textovým súborom o veľkosti 10688 bajtov, pridávam obrázky súboru pred a po kompresii a porovnávam veľkosť neskomprimovaného a skomprimovaného súboru (4976 bajtov). Ten sa mi kompresným algoritmom podarilo zmenšiť na 46,5% pôvodnej veľkosti aj vďaka tomu, že sa text v súbore niekoľkokrát opakoval. Potom popisujem návod ako manipulovať s príslušnou funkciou, ktorú som použil. Dodatočne pridávam ukážku zdrojového kódu dekompresie. Taktiež aj pri metóde LZW simulujem kompresiu a dekompresiu stringu, pridávam výpis kompresie, výpis slovníka – ASCII znaky plus nové reťazce a prevod dekompresie. Pridávam ukážky z prevodových algoritmov.

Ďalší vývoj tejto témy by som videl v hlbšom porovnaní novších obdôb kompresnej metódy LZ77 ako napríklad algoritmus DEFLATE a jeho variant (Deflate64, Enhanced Deflate), ktorý používa kombináciu algoritmu LZ77 a Huffmanove kódovanie.

Literatúra

[1] VEČERKA, Arnošt. Komrese dat [online]. [2016-04-25]. Dostupné z URL: https://phoenix.inf.upol.cz/esf/ucebni/komprese.pdf

[2] TAGALOVÁ, Lenka; Malenko, Jaromír. Algorimy komprese dat. SISAL MFF UK, 1996. [online]. [2016-04-25]. Dostupné z URL: http://www.ms.mff.cuni.cz/~malej9am/doc/komprese.php?lang=en

[3] WELCH, Terry. A Technique for High-Performance Data Compression. IEEE Computer. 1984.

[4] DIPPERSTEIN, Michael. Lempel-Ziv-Welch (LZW) Encoding Discussion and Implementation [online]. [2016-04-25]. Dostupné z URL: http://michael.dipperstein.com/lzw/

[5] About GNU Octave. [online]. [2016-04-24]. Dostupné z URL: http://www.gnu.org/software/octave/about.html

[6] Stránka Octave-Forge: Packages. [online]. [2016-04-24]. Dostupný z URL: http://octave.sourceforge.net/packages.php.

[7] NELSON, Mark. LZW Data Compression on Dr. Dobbs Journal. 1989. [online]. [2016-04-24]. Dostupné z URL: http://marknelson.us/1989/10/01/lzw-data-compression/

[8] STORER, James A.; Szymanshi, Thomas G.. Data Compression via Textual Substitution. Journal of the ACM. 1982. 951 s.

[9] Data Compression Reference Center: RASIP working group. The LZ77 algorithm. Faculty of Electrical Engineering and Computig, University of Zagreb. 1997.

[10] Data Compression Reference Center: RASIP working group. The LZ78 algorithm. Faculty of Electrical Engineering and Computig, University of Zagreb. 1997.

[11] BENEŠ, Miroslav. Komprese dat. Katedra informatiky FEI VŠB-TUO. [online]. [2016-04-26]. Dostupné z URL: http://www.cs.vsb.cz/benes/vyuka/pte/texty/komprese/ch02s04.html

[12] EBRAHIMI, Samira. LZ77. 2009. [online]. [2016-04-20]. Dostupné z URL: http://www.mathworks.com/matlabcentral/fileexchange/22438-lz77

[13] RIDINO, Giuseppe. LZW Compression Algorithm. 2004. [online]. [2016-04-20]. Dostupné z URL: http://www.mathworks.com/matlabcentral/fileexchange/4899-lzw-compression-algorithm

[14] Frequently asked questions for GNU Octave users. [online]. [2016-04-22]. Dostupné z URL: http://wiki.octave.org/FAQ

[15] EATON, John W.. GNU Octave manual: Installing and Removing Packages. [online]. [2016-04-22]. Dostupné z URL: https://www.gnu.org/software/octave/doc/v4.0.1/Installing-and-Removing-Packages.html

Prílohy

Súčasťou originálu bakalárskej práce je CDROM obsahujúci skriptovacie súbory s implementovanými algoritmami[12][13].

Obsah CD:

Adresár LZ77 obsahuje skriptovací súbor mainLZ77

Adresár LZW obsahuje funkcie: norm2lzw

lzw2norm

lzw_demo1